私のTDD

藤村大介 / @ffu_

TDDとは

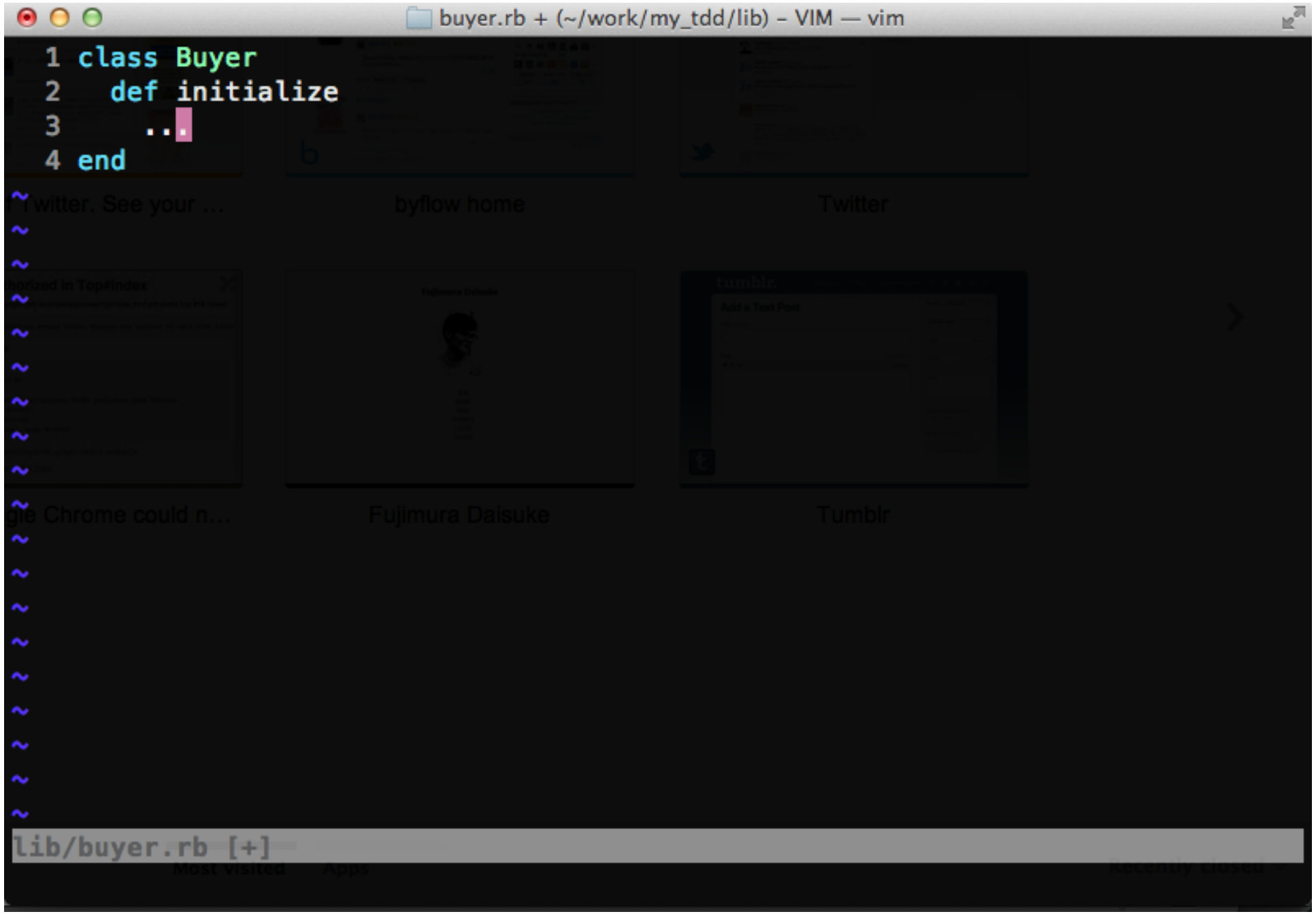
"プログラムに必要な各機能について、最初にテストを書き（これをテストファーストと言う）、そのテストが動作する必要最低限な実装をとりあえず行った後、コードを洗練させる、という短い工程を繰り返すスタイル”

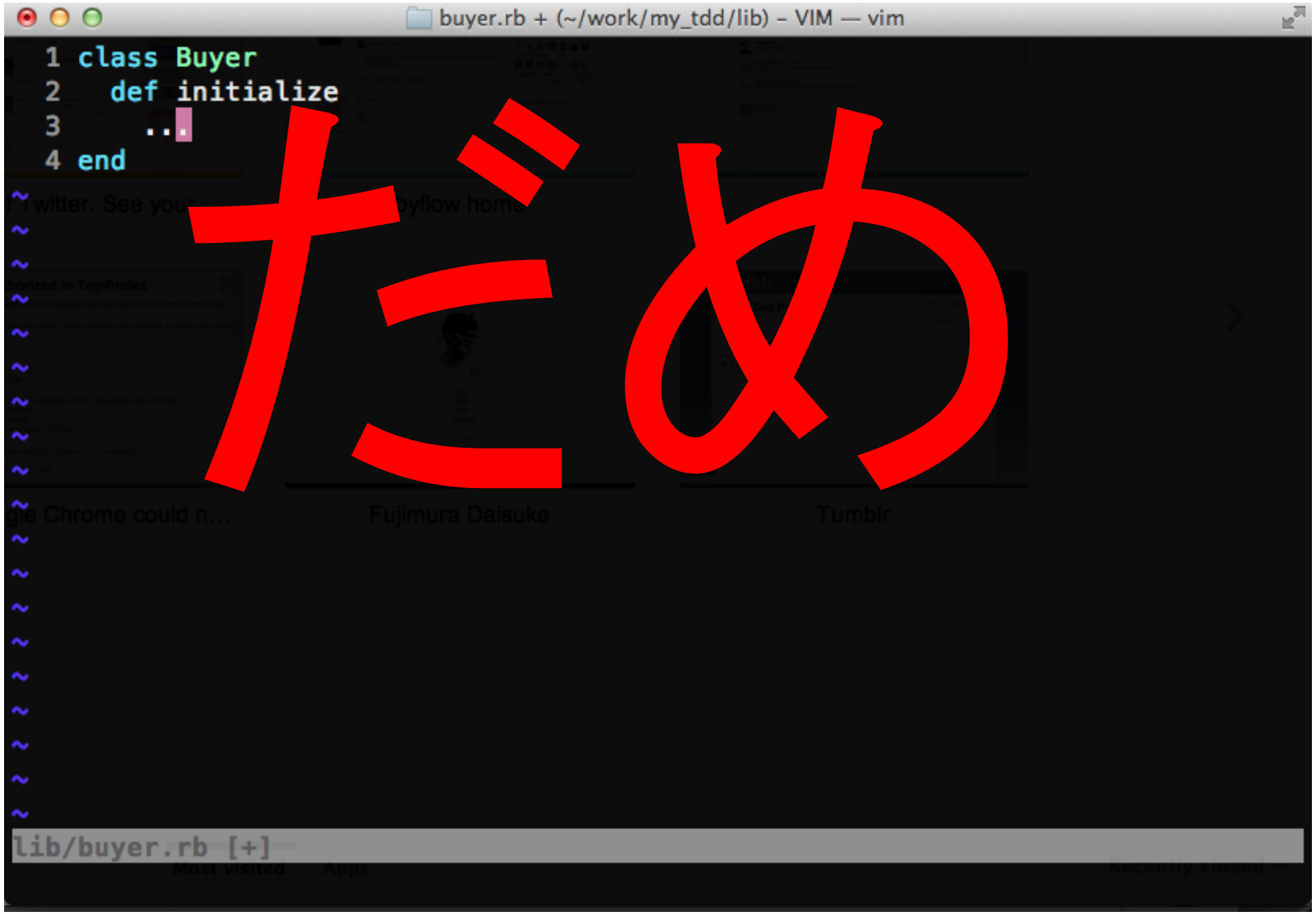
http://ja.wikipedia.org/wiki/テスト駆動開発

実演してみます

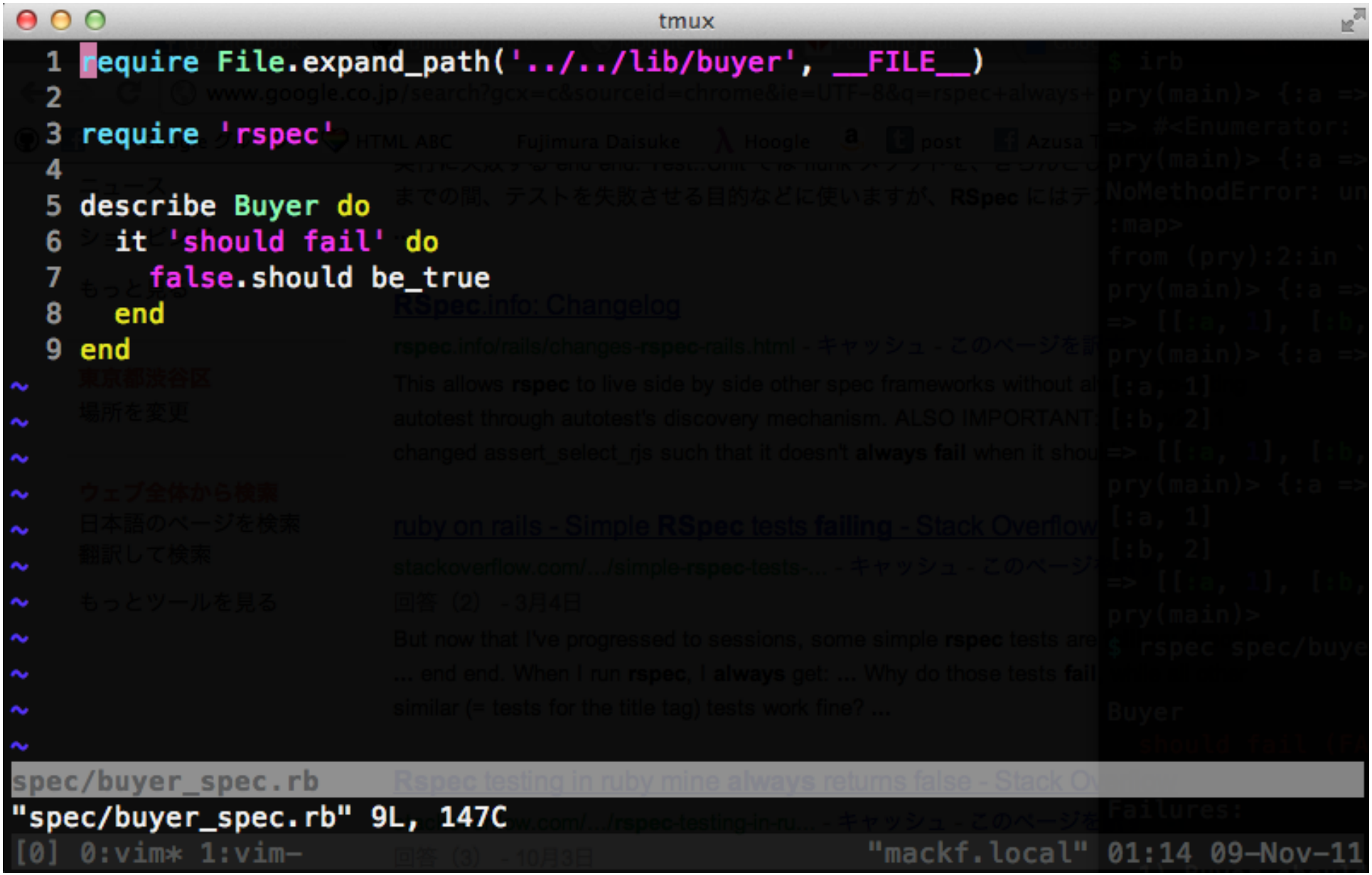
レコードバイヤーを実装します

- 最初に予算をもらう
- レコードは予算があるだけ買える

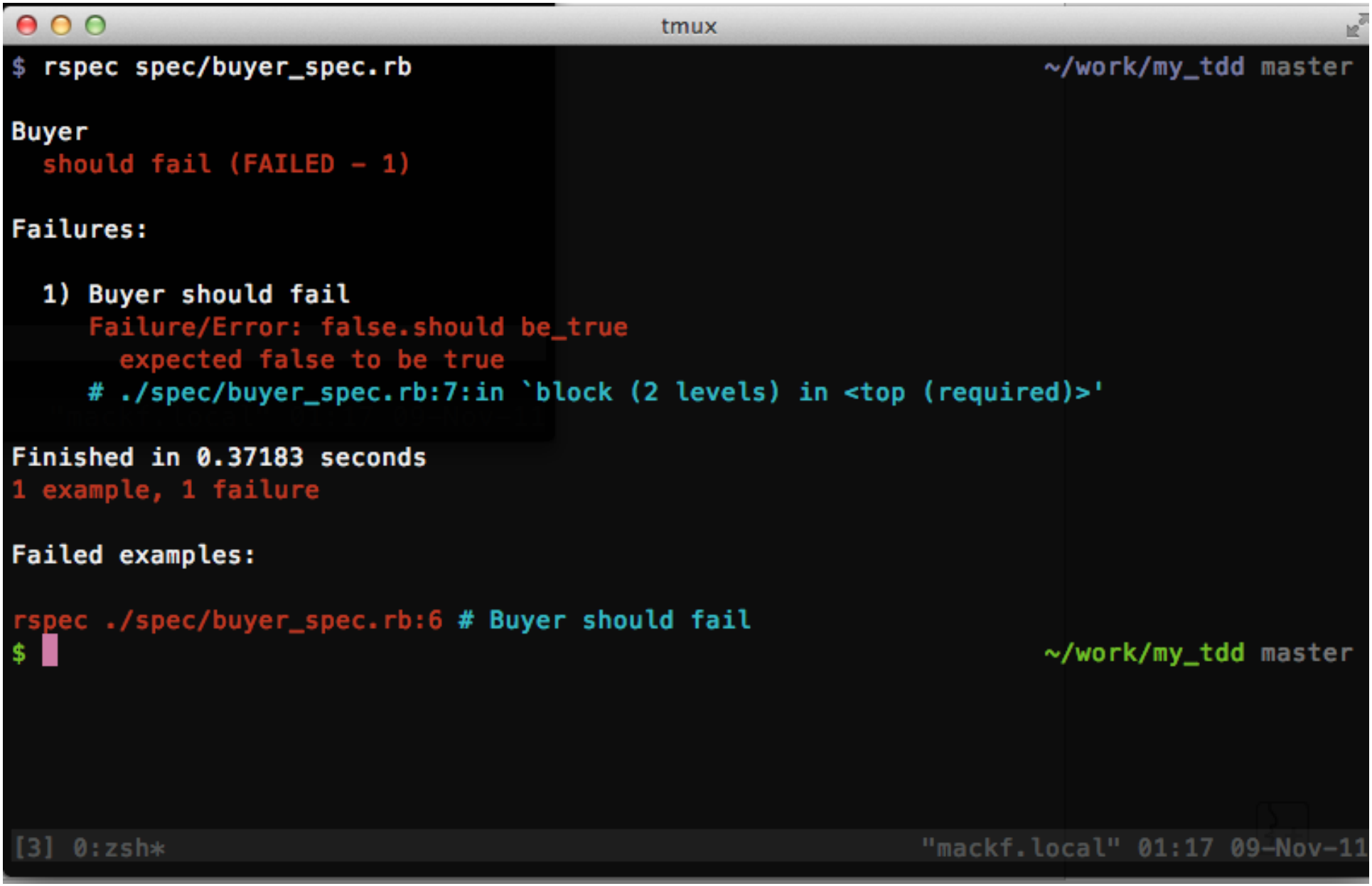

buyer.rb + (~/work/my_tdd/lib) - VIM — vim
1 class Buyer
2 def initialize
3 ...
4 end
lib/buyer.rb [+]


buyer.rb + (~/work/my_tdd/lib) - VIM — vim
1 class Buyer
2 def initialize
3 ...
4 end
~
lib/buyer.rb [+]
だめ

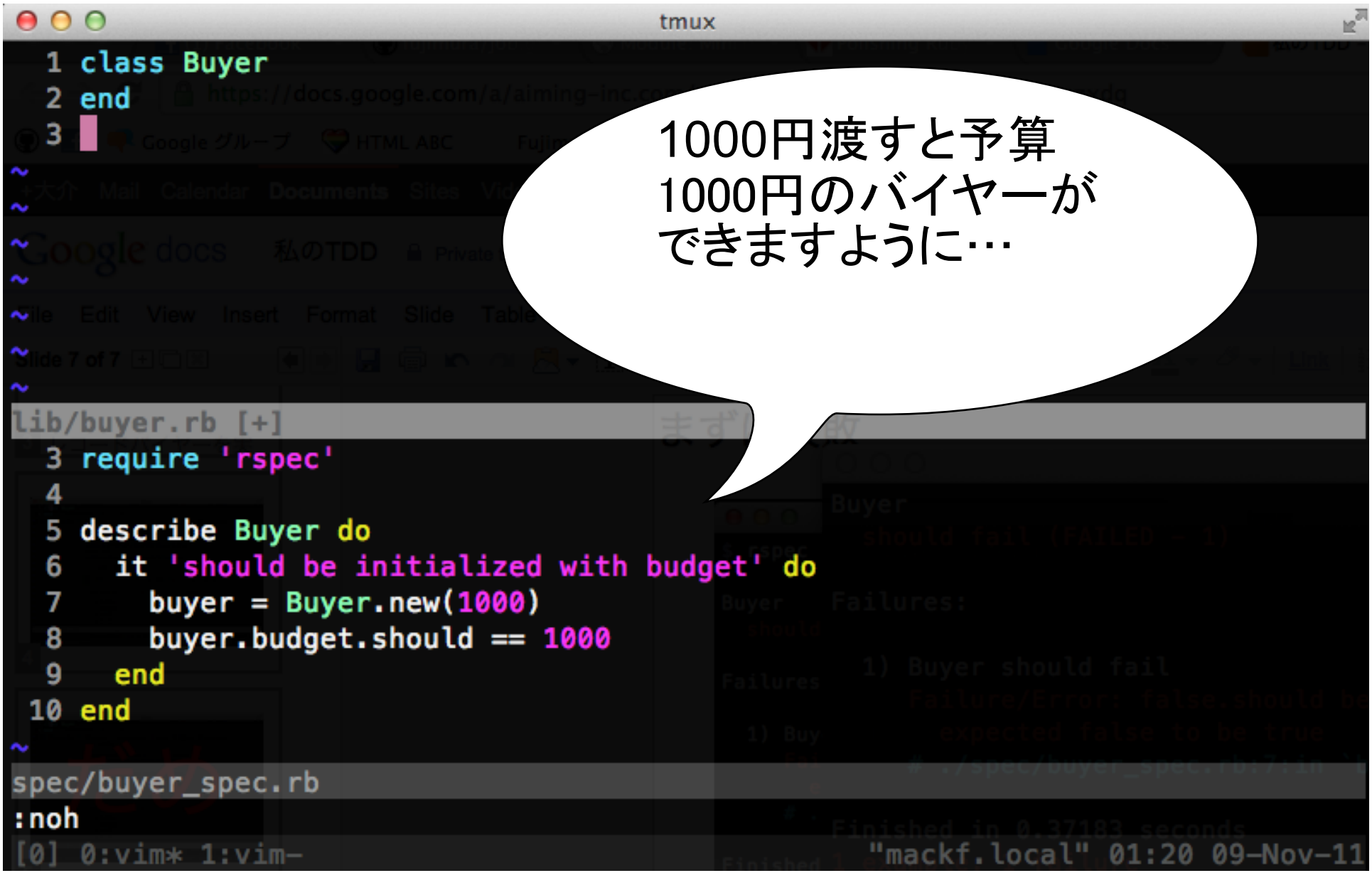
なんでもいいのでまずはテストを書いて、



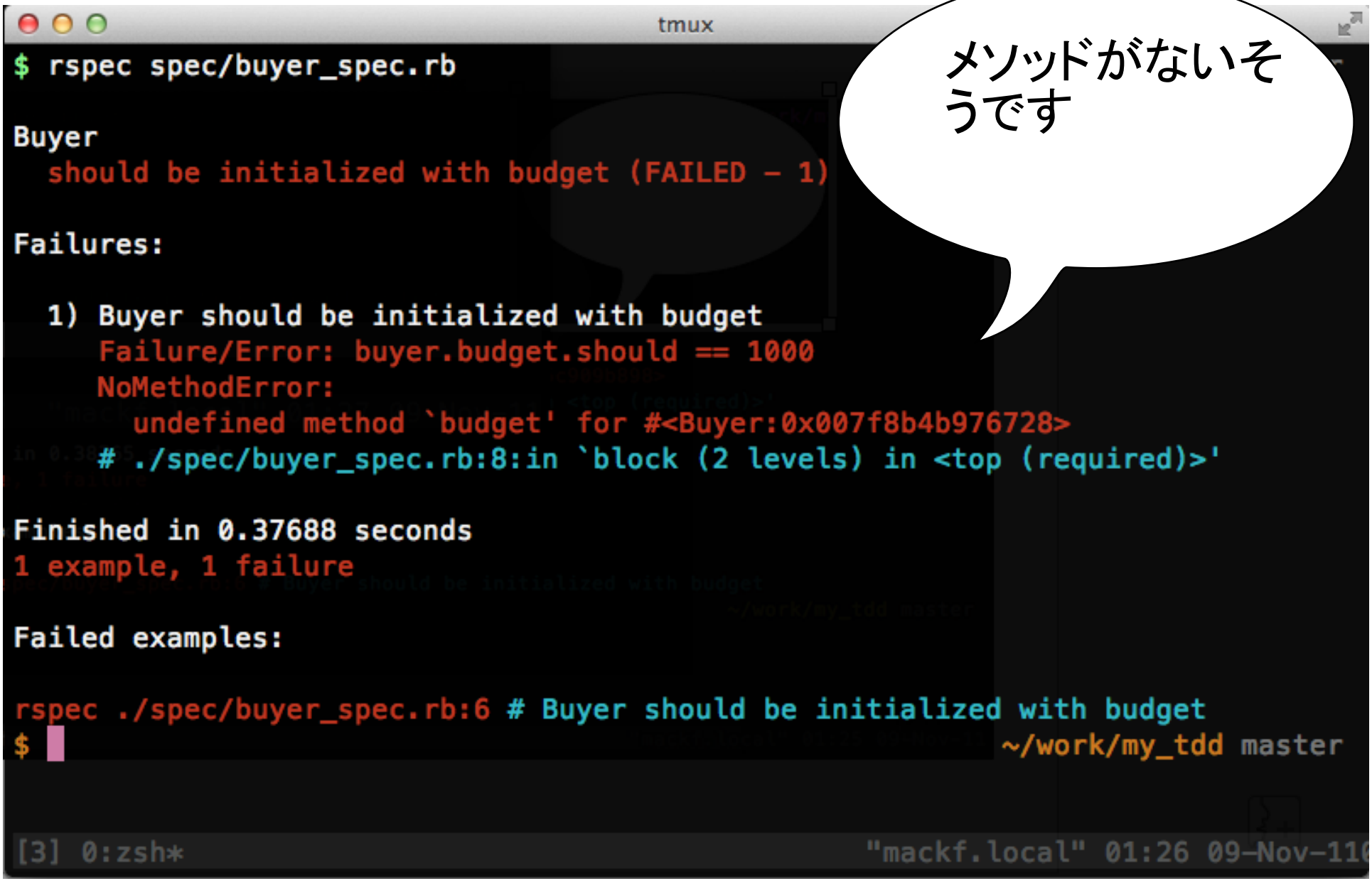
失敗させる（儀式）


tmux
$ rspec spec/buyer_spec.rb
~/work/my_tdd master
Buyer
should fail (FAILED - 1)
Failures:
1) Buyer should fail
Failure/Error: false.should be_true
expected false to be true
./spec/buyer_spec.rb:7:in `block (2 levels) in <top (required)>'
Finished in 0.37183 seconds
1 example, 1 failure
Failed examples:
rspec ./spec/buyer_spec.rb:6 # Buyer should fail
$
~/work/my_tdd master
[3] 0:zsh*
"mackf.local" 01:17 09-Nov-11

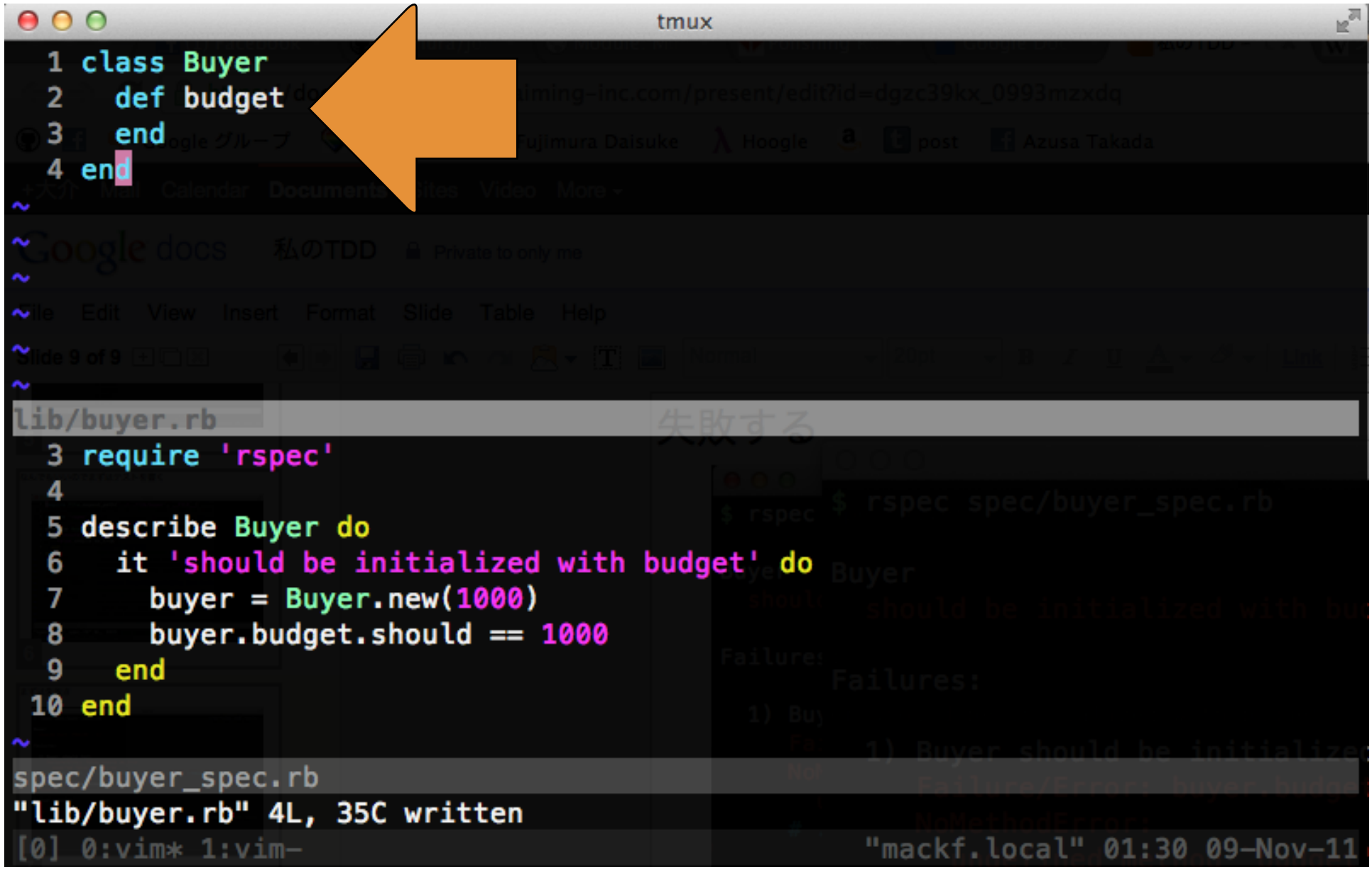
それから、希望する挙動を示すコードをテストとして書く



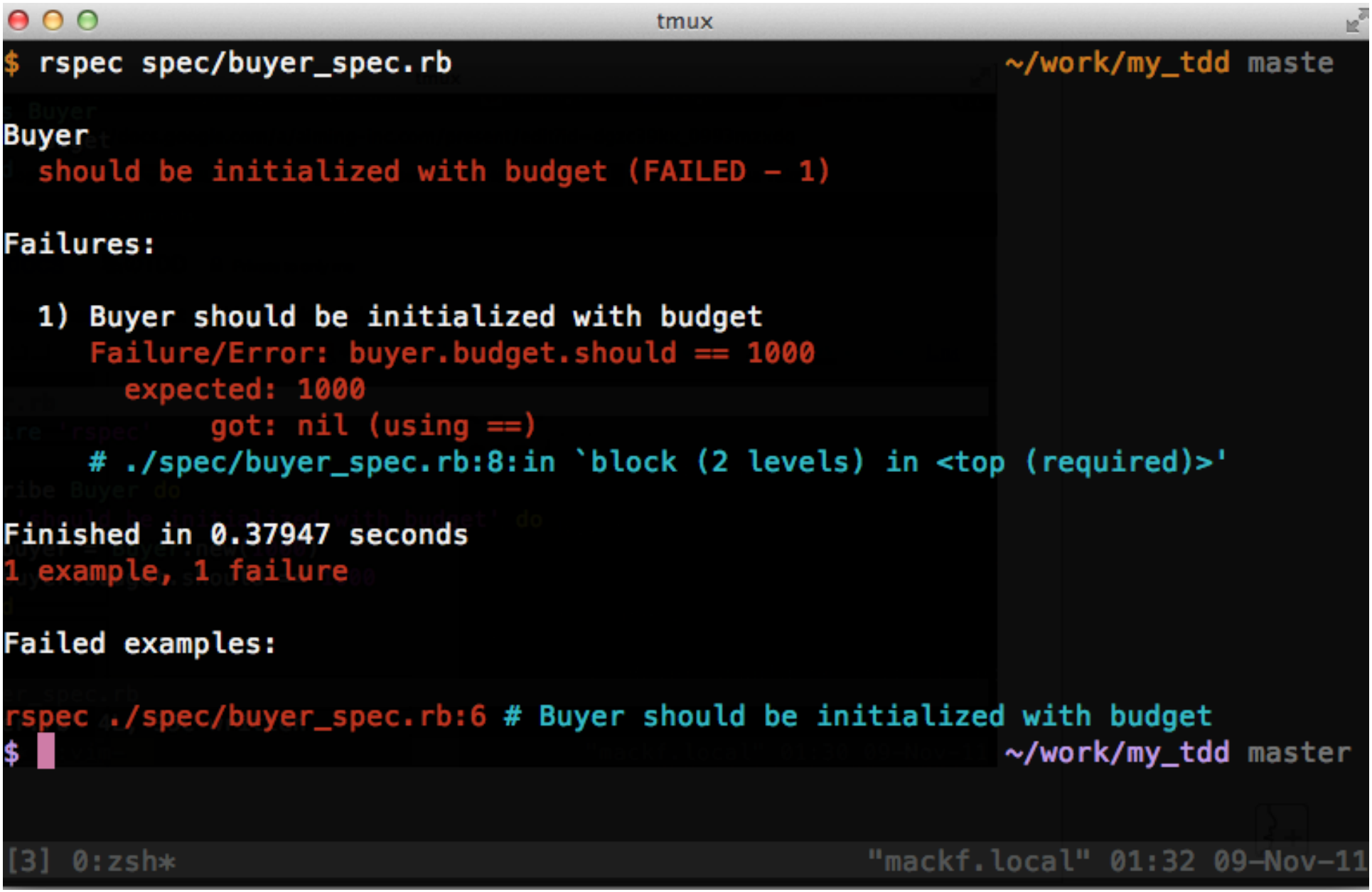
失敗する


tmux
$ rspec spec/buyer_spec.rb
Buyer
should be initialized with budget (FAILED - 1)
Failures:
1) Buyer should be initialized with budget
Failure/Error: buyer.budget.should == 1000
NoMethodError:
undefined method `budget' for #<Buyer:0x007f8b4b976728>
./spec/buyer_spec.rb:8:in `block (2 levels) in <top (required)>'
Finished in 0.37688 seconds
1 example, 1 failure
Failed examples:
rspec ./spec/buyer_spec.rb:6 # Buyer should be initialized with budget
$
~/work/my_tdd master
[3] 0:zsh*
"mackf.local" 01:26 09-Nov-11
メソッドがないそうです

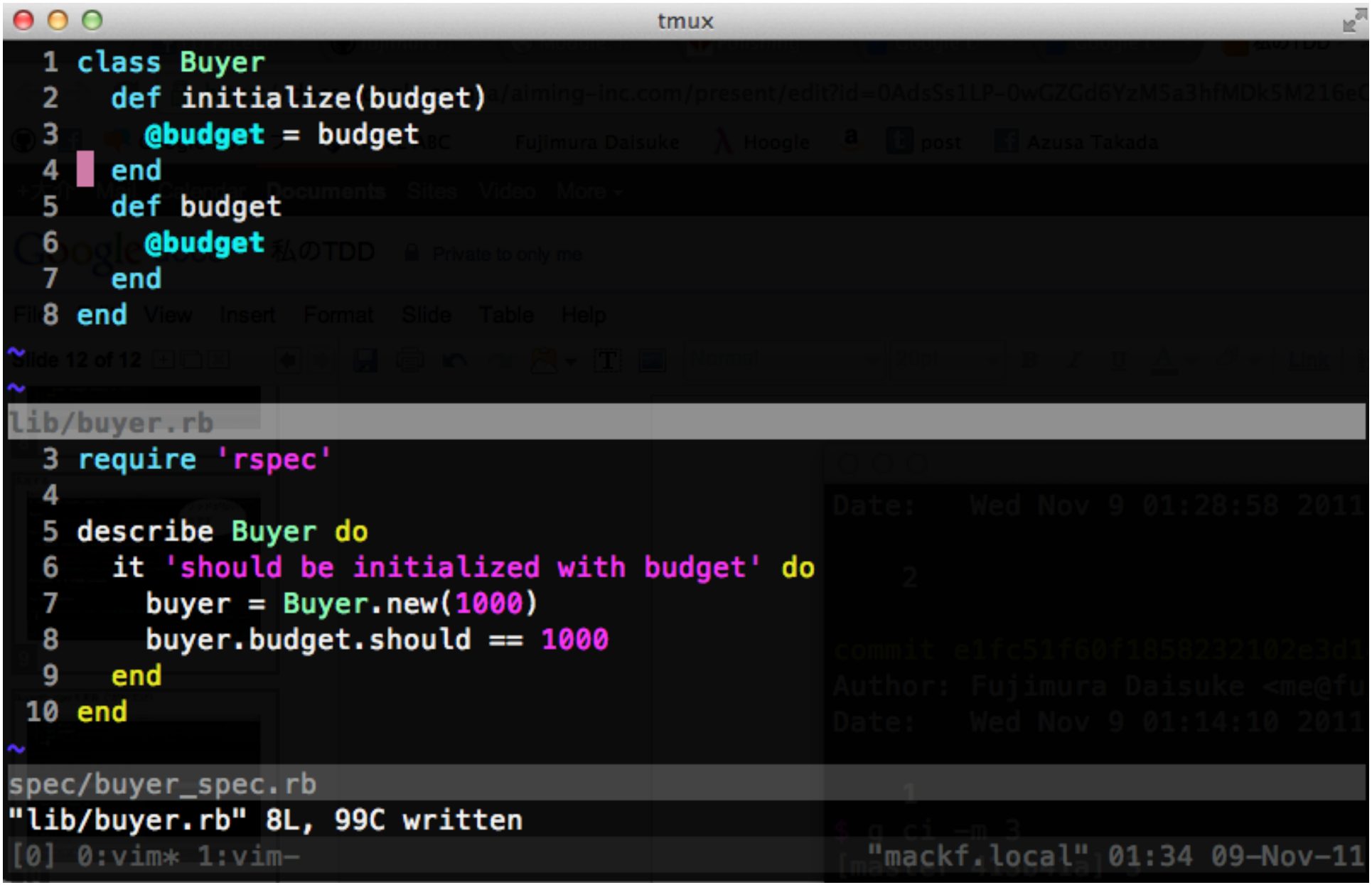
Buyer#budget を実装（”がわ”だけ）



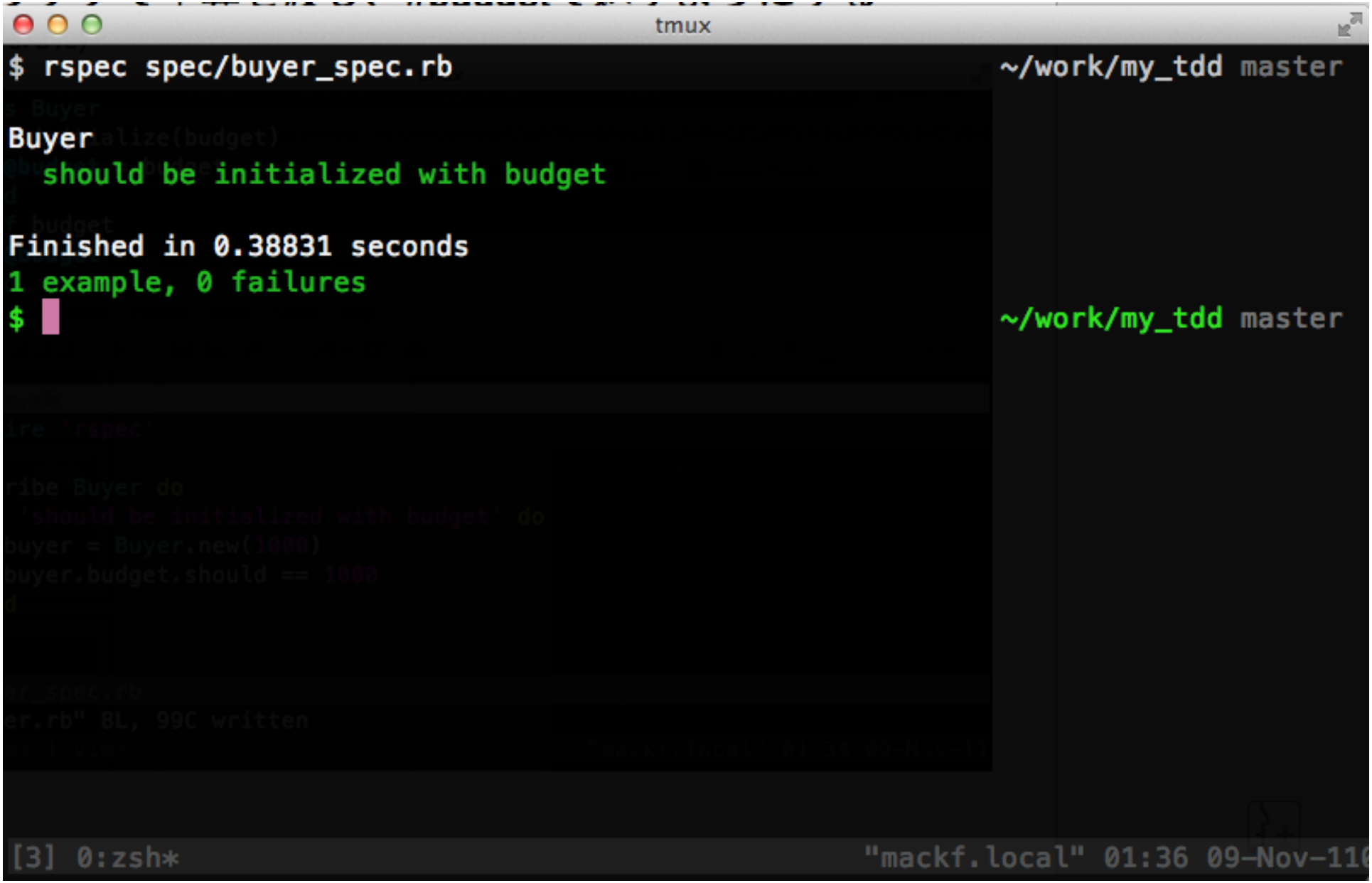
失敗する 1000が欲しかったのにnilが返りました（当たり前）



コンストラクタで予算を渡し、#budgetで返すようにする
(ちょっと端折りました)



できました


tmux
$ rspec spec/buyer_spec.rb
~/work/my_tdd master
Buyer
should be initialized with budget
Finished in 0.38831 seconds
1 example, 0 failures
$
~/work/my_tdd master
[3] 0:zsh*
"mackf.local" 01:36 09-Nov-11

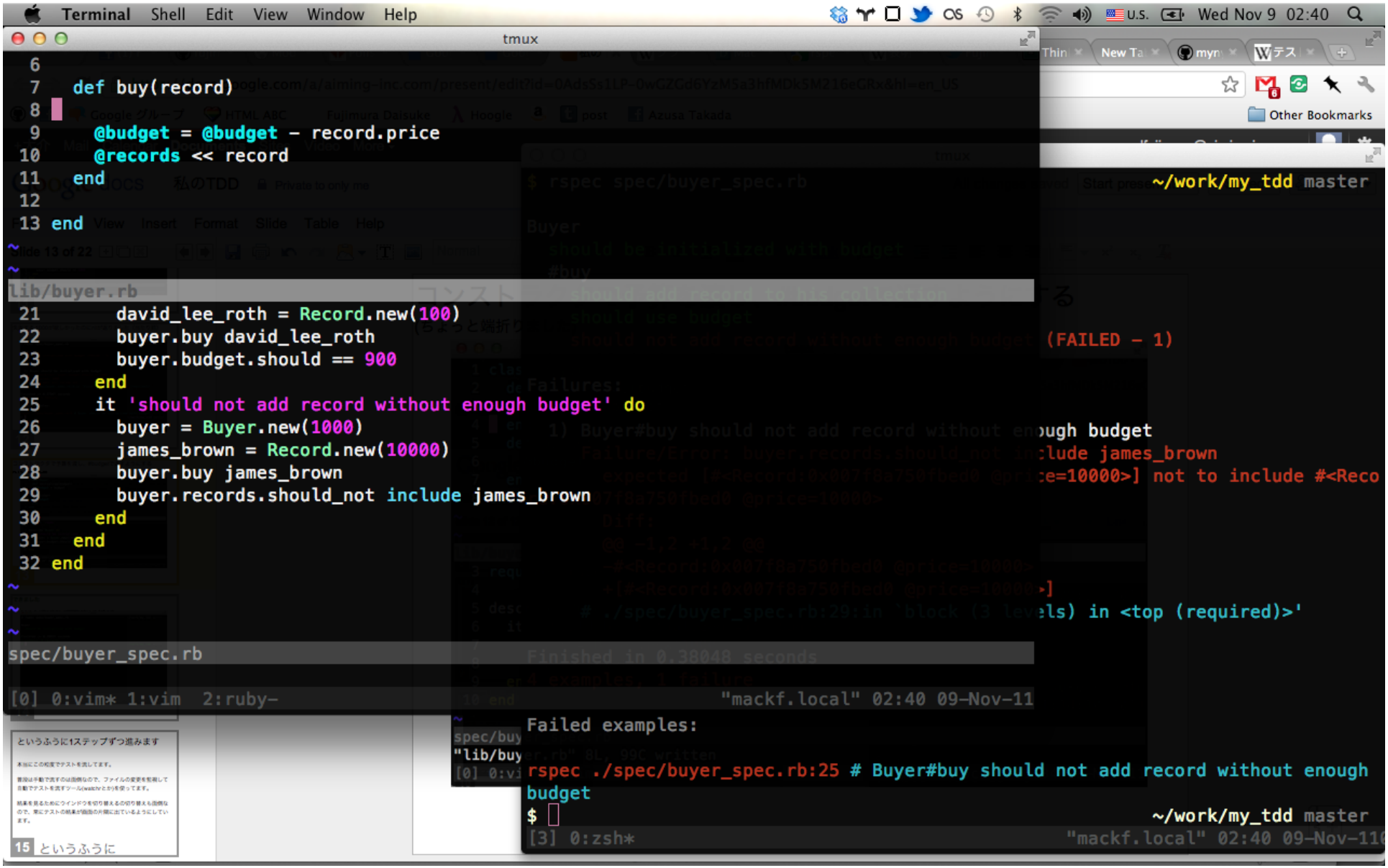
というふうに

最小の粒度でテストを流して開発していきます。

普段は手動で流すのは面倒なので、ファイルの変更を監視して自動でテストを流すツール(watchrとか)を使ってます。

結果を見るためにウインドウを切り替えるのも面倒なので、常にテストの結果が画面の片隅に出ているようにしています。

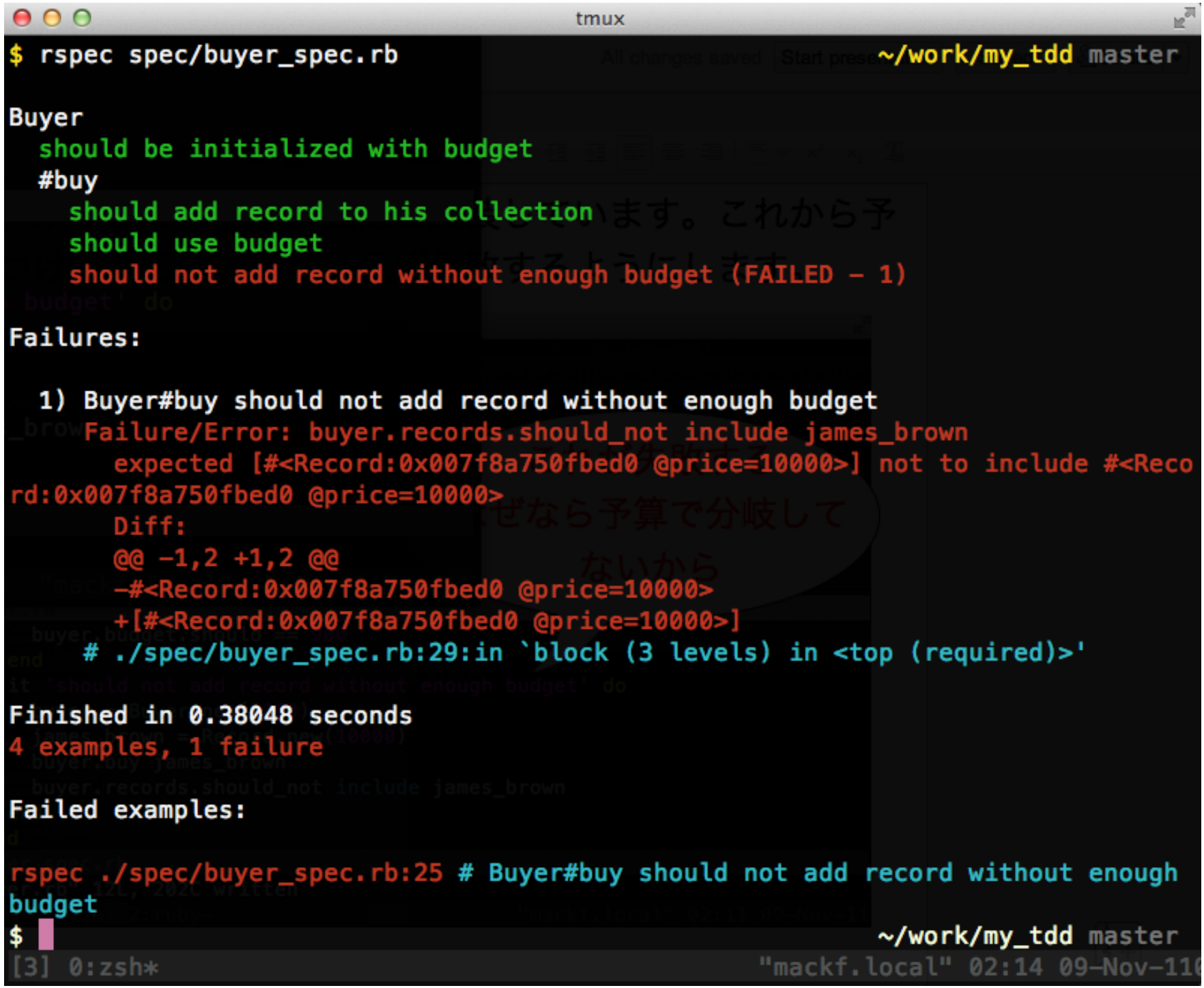
ノートPCの場合はこんな感じでターミナルを重ねている



時は流れ

現在レコード購入機能を実装しています。これから予算制限をつけるところ。（今は無制限に買える）まずはテストを書きます

もちろん
まず失敗
させる



実装簡単だし、
いちいち失敗させるのはアホらしいと思いますよね？
しかし、

条件分岐を実装する際は、まずすべての分岐を通るテストを書きます

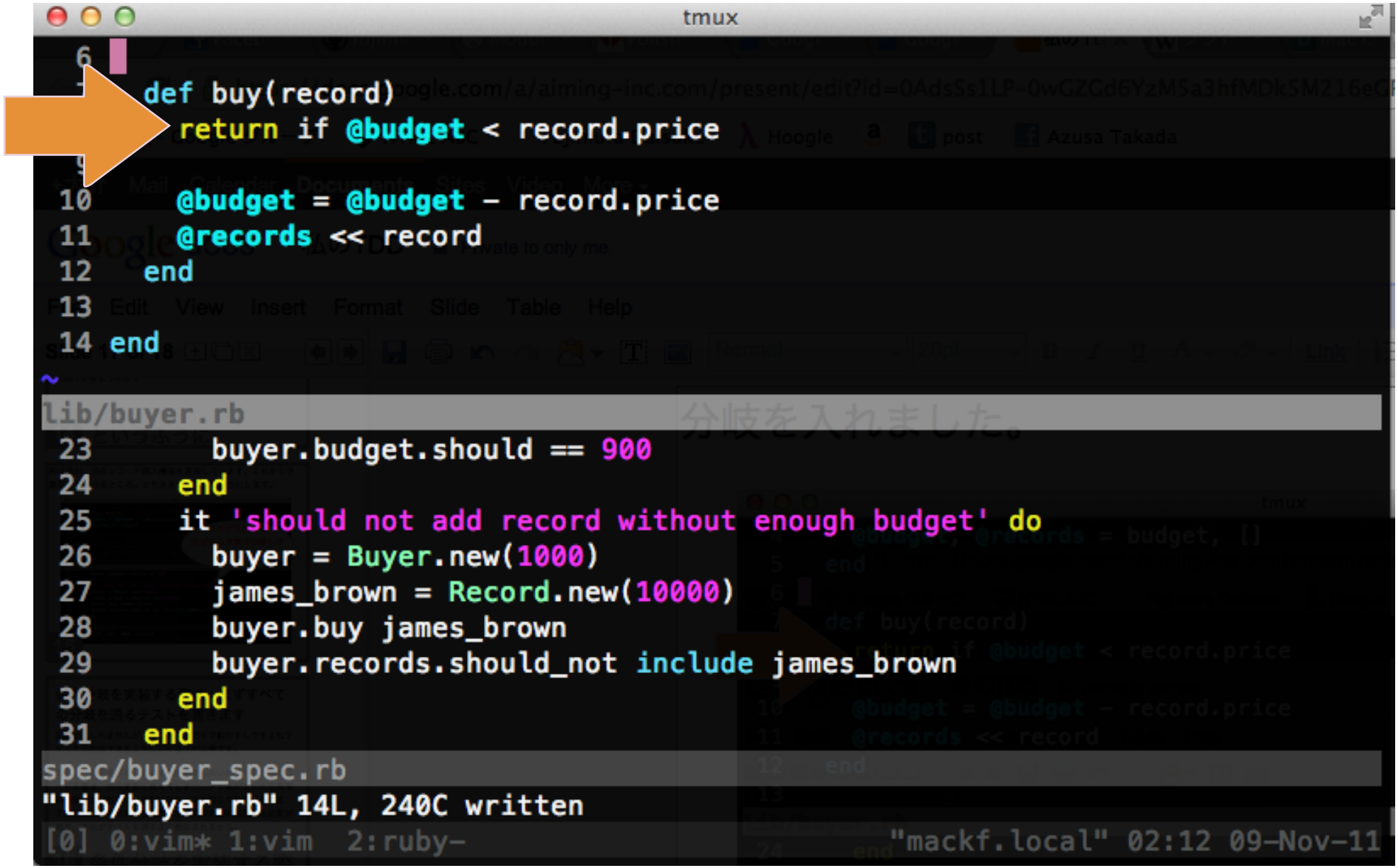
面倒かもしれませんが、どうせあとで全部手で動かすんですよね？だったら再現できるようにしたほうがお得です。

さらに、正常系（もしくはその時動かしたい分岐）のみ通るコードを書き、他の実装に進んでしまい…アッ気がついたら異常系の想定が漏れてた、みたいなケースも起こりにくくなります。なぜならテストを書く際に全パスの処理を想定する必要があり、それを実装するようテストが示してくれるから。

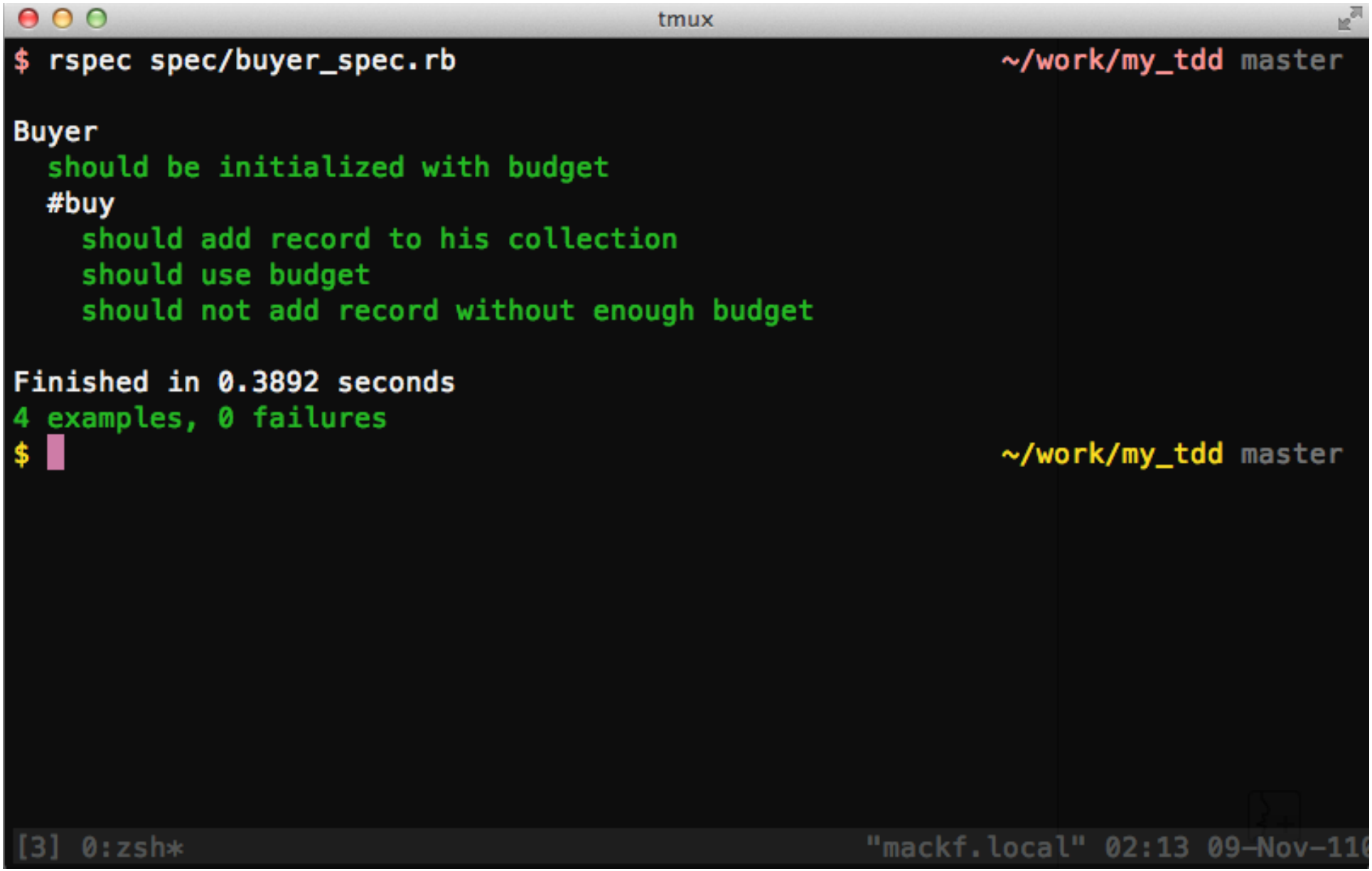
（本当にやってます）

実装に戻ります

分岐を入れました。


tmux
def buy(record)
return if @budget < record.price
@budget = @budget - record.price
@records << record
end
end
lib/buyer.rb
buyer.budget.should == 900
end
it 'should not add record without enough budget' do
buyer = Buyer.new(1000)
james_brown = Record.new(10000)
buyer.buy james_brown
buyer.records.should_not include james_brown
end
end
spec/buyer_spec.rb
"lib/buyer.rb" 14L, 240C written
[0] 0:vim* 1:vim 2:ruby-
"mackf.local" 02:12 09-Nov-11

通りました これで安心の条件分岐が実現された。



これを少しずつ進めていきます

ちなみにソフトウェアテストそのものの技法をしっかり身につけておくとテストは書きやすくなり、また堅牢なコードになると思います。

この本がおすすめ

はじめて学ぶソフトウェアのテスト技法
リー・コープランド（著）, 宗 雅彦（翻訳）

最後に、私のTDD

私、ソフトウェア開発の技法の中で何が好きか、と言うと、断然TDDです。夢中とまでは言いませんが、今更離れるのはあまりに辛いツールであります。

そんなTDDとは私にとって一体何なのだろうのか？の話。

私は面倒くさがり、しかも仕事が雑です

更に落ち着きがなく、極めて散漫な性格です。仕事に時間がかかるのも嫌いです。

だからこそのTDD。最初はウザかったけど、

- 手戻りがない（常に再帰テストされる）
- ミスが少なくなる（事前に全分岐のテストケースを想定）
- やるべきことが示される（まず失敗させて、それから実装）
- 必要ないことをやらないで済む（テストが通れば実装完了）

と、気がついたらTDDは自分の弱点をそっとカバーしてくれるのでした。

みなさんも騙されたと思ってやってみてください。

最初は面倒かもしれません。慣れは必要です。

ツールや環境を整備すると、かなり負担は減ります。
開発環境でCIを動かす感覚で常にテストを流しておくとかなり楽です。
watchrあたりを活用するとよいです。

watchr:
https://github.com/mynyml/watchr

終わり